

PTO/SB/21 (08-03)

Approved for use through 08/30/2003. OMB 0651-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMITTAL FORM (to be used for all correspondence after initial filing)	Application Number	09/710,948	
	Filing Date	11/13/2000	
	First Named Inventor	Samsikrishna Vamsi Krishna	
	Art Unit	2122	
	Examiner Name	Kuo Liang J. Tang	
Total Number of Pages in This Submission	53	Attorney Docket Number	JP920000282US1

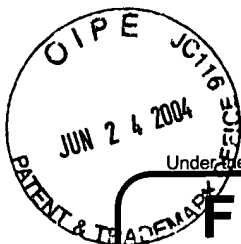
ENCLOSURES (Check all that apply)		
<input checked="" type="checkbox"/> Fee Transmittal Form	<input type="checkbox"/> Drawing(s)	<input type="checkbox"/> After Allowance communication to Technology Center (TC)
<input checked="" type="checkbox"/> Fee Attached	<input type="checkbox"/> Licensing-related Papers	<input type="checkbox"/> Appeal Communication to Board of Appeals and Interferences
<input type="checkbox"/> Amendment/Reply	<input type="checkbox"/> Petition	<input checked="" type="checkbox"/> Appeal Communication to TC (Appeal Notice, Brief, Reply Brief*)
<input type="checkbox"/> After Final	<input type="checkbox"/> Petition to Convert to a Provisional Application	<input type="checkbox"/> Proprietary Information
<input type="checkbox"/> Affidavits/declaration(s)	<input type="checkbox"/> Power of Attorney, Revocation	<input type="checkbox"/> Status Letter
<input type="checkbox"/> Extension of Time Request	<input type="checkbox"/> Change of Correspondence Address	<input checked="" type="checkbox"/> Other Enclosure(s) (please identify below):
<input type="checkbox"/> Express Abandonment Request	<input type="checkbox"/> Terminal Disclaimer	Acknowledgement postcard
<input type="checkbox"/> Information Disclosure Statement	<input type="checkbox"/> Request for Refund	
<input type="checkbox"/> Certified Copy of Priority Document(s)	<input type="checkbox"/> CD, Number of CD(s) _____	
<input type="checkbox"/> Response to Missing Parts/Incomplete Application	Remarks *in triplicate (Appeal Brief <u>11</u> pages, Appendix "AA" <u>6</u> pages)	
<input type="checkbox"/> Response to Missing Parts under 37 CFR 1.52 or 1.53		

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT	
Firm or Individual name	Anthony V.S. England
Signature	<i>Anthony V.S. England</i>
Date	6-21-2004

CERTIFICATE OF TRANSMISSION/MAILING		
I hereby certify that this correspondence is being facsimile transmitted to the USPTO or deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date shown below.		
Typed or printed name	Anthony V.S. England	
Signature	<i>Anthony V.S. England</i>	Date
		6-21-2004

This collection of information is required by 37 CFR 1.5. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.



PTO/SB/17 (10-03)

Approved for use through 07/31/2006. OMB 0651-0032

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2004

Effective 10/01/2003. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT

(\$) 330.00

Complete if Known

Application Number	09/710,948
Filing Date	11/13/2000
First Named Inventor	Samsikrishna Vamsi Krishna
Examiner Name	Kuo Liang J. Tang
Art Unit	2122
Attorney Docket No.	JP920000282US1

METHOD OF PAYMENT (check all that apply)☐ Check ☐ Credit card ☐ Money Order ☐ Other ☐ None☒ Deposit Account:Deposit Account Number
Deposit Account Name

09-0457

International Business Ma

The Director is authorized to: (check all that apply)

☒ Charge fee(s) indicated below ☒ Credit any overpayments☒ Charge any additional fee(s) or any underpayment of fee(s)☐ Charge fee(s) indicated below, except for the filing fee to the above-identified deposit account.**FEE CALCULATION****1. BASIC FILING FEE**

Large Entity		Small Entity		Fee Description	Fee Paid
Fee Code	Fee (\$)	Fee Code	Fee (\$)		
1001	770	2001	385	Utility filing fee	
1002	340	2002	170	Design filing fee	
1003	530	2003	265	Plant filing fee	
1004	770	2004	385	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	
SUBTOTAL (1)					(\$) 0

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

		Extra Claims	Fee from below	Fee Paid
Total Claims		-20** =	X	
Independent Claims		-3** =	X	
Multiple Dependent				

Large Entity		Small Entity		Fee Description
Fee Code	Fee (\$)	Fee Code	Fee (\$)	
1202	18	2202	9	Claims in excess of 20
1201	86	2201	43	Independent claims in excess of 3
1203	290	2203	145	Multiple dependent claim, if not paid
1204	86	2204	43	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2)

(\$) 0

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)**3. ADDITIONAL FEES**

Large Entity		Small Entity		Fee Description	Fee Paid
Fee Code	Fee (\$)	Fee Code	Fee (\$)		
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for <i>ex parte</i> reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	420	2252	210	Extension for reply within second month	
1253	950	2253	475	Extension for reply within third month	
1254	1,480	2254	740	Extension for reply within fourth month	
1255	2,010	2255	1,005	Extension for reply within fifth month	
1401	330	2401	165	Notice of Appeal	
1402	330	2402	165	Filing a brief in support of an appeal	\$330.00
1403	290	2403	145	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,330	2453	665	Petition to revive - unintentional	
1501	1,330	2501	665	Utility issue fee (or reissue)	
1502	480	2502	240	Design issue fee	
1503	640	2503	320	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	770	2809	385	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	770	2810	385	For each additional invention to be examined (37 CFR 1.129(b))	
1801	770	2801	385	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify)

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) (\$) 330.00

SUBMITTED BY

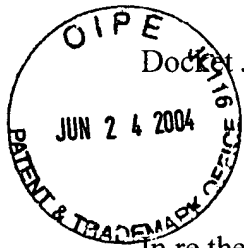
(Complete if applicable)

Name (Print/Type)	Anthony V.S. England	Registration No. (Attorney/Agent)	35,129	Telephone	512-477-7165
Signature	Anthony V.S. England	Date	6-21-2004		

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

This collection of information is required by 37 CFR 1.17 and 1.27. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.



Docket JP920000282

Appl. No. 09/710,948

February 14, 2001

AF/2122
\$
IFW

In the United States Patent and Trademark Office

In re the application of:

Krishna et al.

Filed: 11/13/2000

For: A Method for Global
Breakpoint Identification

Appl. No.: 09/710,948

Applicant's Docket:
JP920000282US1

Group Art Unit: 2122

Examiner: Kuo Liang J. Tang

APPEAL BRIEF

Dear Sir:

REAL PARTY IN INTEREST

The assignee, International Business Machines Corporation, is the real party in interest.

RELATED APPEALS AND INTERFERENCES

This is the first appeal in the present patent application. There are no other appeals or interferences known to the appellant or its legal representative. International Business Machines Corporation is the sole assignee of the patent application.

06/28/2004 YPOLITE1 00000143 090457 09710948

01 FC:1402 330.00 DA

STATUS OF CLAIMS

Claims 1-39 were originally presented in the application.

In a first Office action dated August 4, 2003 (the "First Office Action"), claims 3, 16 and 19 were rejected under 35 USC 112, second paragraph, as being indefinite. Claims 1-39 were rejected under 35 USC 103(a).

In response to the First Office Action certain ones of the claims were amended in a reply of November 4, 2003, so that the claims more clearly point out patentable distinctions with regard to the art cited and to correct indefiniteness. Specifically, claim 1 and its counterpart independent claims, 14 and 27, were amended, along with dependent claims 3, 16 and 29. Claims 6, 19 and 32 were canceled.

Claims 1-5, 7-18, 20-31 and 33-39 were finally rejected in an Office action of January 14, 2004 (the "Final Office Action") under 35 USC 103(a). The rejections under 35 USC 112, second paragraph, were overcome by the amendments submitted in reply to the First Office Action. Appellant has appealed from the final rejection of the claims.¹ No claims have been allowed.

STATUS OF AMENDMENTS

Claims 1-5, 7-18, 20-31 and 33-39 are pending in the case. No amendments have been submitted after the Final Office Action. The claims set out in Appendix "AA" herein below reflect the status of the claims as entered responsive to Appellant's reply of November 4, 2003.

SUMMARY OF INVENTION

The present invention is claimed in the form of a method, a system, and a computer program product in independent claims 1, 14 and 27, respectively.

Claim 1 sets out a method of identifying a global breakpoint for debugging computer software that includes receiving a file name for an executable image file, wherein the executable image file is loaded in memory of a computer system and the global breakpoint is to be placed in the image for executing by the computer system.²

¹ Notice of Appeal, received by USPTO on April 19, 2004.

² Present application, FIG. 1; page 7, line 17 - page 8, line 2, (describing how the executable relates to the actual physical memory of a target system); see also page 10, line 25 - page 11, line 3 (describing how the memory location is identified when an exception is triggered by a break point instruction at that location).

The method also includes receiving a symbol expression for a location in the executable image file where the global breakpoint is to be placed and passing the symbol expression and the file name to a first operating system module running on the computer system.³

Also, according to the claim, the file name for the executable image file is passed to a second operating system module and a file offset and file identifier for the executable image are returned by the modules and represented *in* the executable, i.e., inserted into the executable image file.⁴

Independent claims 14 and 27 have similar language, each according to the respective forms of the invention they claim.

This claimed arrangement is advantageous for a number of reasons. It is advantageous that the executable image can be debugged using operating system modules of a computer system that are the same modules available in the target computer system. That is, the modules are *not* special modules of debugger software.⁵ Also, the arrangement allows identifying a breakpoint without additional debug information at run time and without making any assumptions about load location or process contexts of executable code being debugged.⁶

According to claim 2, the file identifier is a file name.⁷ Claims 15 and 28 have similar language.

According to claim 3, the file identifier is an inode of a Unix operating system.⁸ Claims 16 and 29 have similar language.

According to claim 4, the file identifier is a file control block of a non-Unix operating system.⁹ Claims 17 and 30 have similar language.

According to claim 5, the method includes resolving a virtual address of the code to the file identifier and the offset.¹⁰ Claims 18 and 31 have similar language.

According to claims 7 and 8, the method includes providing a hash list to look up the global breakpoint using the file identifier and the offset.¹¹ Claims 20, 21, 33 and 34 have similar

³ Present application, FIG. 3; page 9, line 29 through page 10, line 5.

⁴ Present application, FIG. 3; page 9, line 25 through page 10, line 15.

⁵ Present application, page 6, lines 22-24.

⁶ Present application, page 8, lines 5-12.

⁷ Present application, FIG. 3; page 9, lines 29-32.

⁸ Present application, FIG. 2; page 9, lines 1-5.

⁹ Present application, FIG. 3; page 9, line 32- page 10, line 15.

¹⁰ Present application, page 6, lines 6-7.

¹¹ Present application, page 10, lines 17-22.

language.

According to claim 9, the method includes deriving the file identifier and the offset using a virtual address.¹² Claims 22 and 35 have similar language.

According to claim 10, the deriving is dependent upon information maintained by an operating system to map executable files to memory.¹³ Claims 23 and 36 have similar language.

According to claim 11, the method includes determining file identifier and said offset from said virtual address for a memory mapped region.¹⁴ Claims 24 and 37 have similar language.

According to claim 12, two or more virtual addresses exist for the software code.¹⁵ Claims 25 and 38 have similar language.

According to claim 13, the global breakpoint is contained in a private-per-process copy of a physical page of the software code.¹⁶ Claims 26 and 39 have similar language.

ISSUES

Issue 1. Are claims 1-2, 15-16 and 27-28 unpatentable under 35 U.S.C. 103(a) in view of the combination of Rosenberg, Jonathan B, "How Debuggers Work: Algorithms, Data Structures and Architecture" ("Rosenberg") and U.S. Patent 5,881,288 ("Sumi")?

Issue 2. Are claims 3, 16 and 29 unpatentable under 35 U.S.C. 103(a) in view of the combination of Rosenberg, Sumi and U.S. Patent 6,584,582 ("O'Connor")?

Issue 3. Are claims 4, 17 and 30 unpatentable under 35 U.S.C. 103(a) in view of the combination of Rosenberg, Sumi and U.S. Patent 6,256,646 ("Starek")?

GROUPING OF CLAIMS

Solely for the purpose of this appeal, the claims stand or fall together according to the following groups:

Group 1: claims 1-2, 5, 7-15, 18, 20-28, 31 and 33-39;

Group 2: claims 3, 16 and 29; and

Group 3: claims 4, 17 and 30.

¹² See above regarding claim 5.

¹³ Present application, page 10, lines 25-29; FIG. 4; page 11, lines 11-19.

¹⁴ Present application, page 10, line 31 - page 11, line 6; FIG. 4, page 11, lines 21-31.

¹⁵ Present application, FIG. 1; page 7, lines 21-25.

¹⁶ Present application, FIG. 1; page 7, lines 26-30.

ARGUMENT

Issue 1. The Final Office Action rejects claims 1-2, 15-16 and 27-28 on the basis that they are obvious in view of Rosenberg and Sumi. Appellant respectfully disagrees. To establish *prima facie* obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art.¹⁷

According to the present application “the target is not the process being debugged but the code that is being *executed*.”¹⁸ To make more clear the patentable distinction of the present invention, claim 1 and its counterpart independent claims, 14 and 27, were amended in response to the First Office Action. In particular, amended claim 1 states, by way of predicate, that a file name is received “for an executable image file,” that “the executable image file is loaded in memory of a computer system for execution by the computer system” and that “the global breakpoint is to be placed in the image.” Also, the present invention involves debugging the executable image by using operating system modules of a computer system that are the same modules available in the target computer system. That is, the modules are *not* special modules of debugger software. Accordingly, Claim 1 were previously amended to state that a symbol expression and the file name are passed “to a first operating system module running on the computer system” and that the symbol expression is “for a location in the executable image file where the global breakpoint is to be placed.”

The First Office Action acknowledged that Rosenberg’s method for identifying a break point setting involves specifying a file name and line number or file offset in a source file, not in an executable as in the subject of the present application.¹⁹ Also, the Final Office Action acknowledges that Rosenberg does not teach setting a break point in an executable file. Applicant agrees. However, the Final Office Action maintains that Sumi teaches setting a break point in an executable file and that it would have been obvious to combine the two references. Applicant disagrees.

First, Sumi does not even teach setting a breakpoint in an executable file. Sumi concerns

¹⁷ MPEP 2143.03 (citing *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974); *In re Wilson*, 424 F.2d 1382, 1385, 165 USPQ 494, 496 (CCPA 1970)).

¹⁸ Present application, page 1, line 32 through page 2, line 2.

¹⁹ First Office Action, page 2, last paragraph.

a program development system for simulating execution of a program being debugged, as will be described further herein below. In contrast, the present invention claims debugging an executable image in the context of the image's location in a target computer system memory.²⁰ Notably, since the operating system of a target system already has functionality to deal with the memory context, the debugging in the present invention includes using an operating system module that is the same module available in computer systems that will ultimately execute the executable image, i.e., the "target" computer systems.²¹

Sumi points out a problem of a prior art arrangement that is addressed by his invention, stating that "The disadvantage of such debugging apparatuses which improve the efficiency of program development using correspondence between variables and resources lies in their inability to show the user the content of the optimization executed inside the program conversion unit, so that when the program is greatly rewritten by the internal processing of the program conversion unit, it will take the user a great deal of time to comprehend how his/her program has been rewritten."²² The "rewritten program" in the above passage is referred to elsewhere by Sumi as "execution code."²³

While the above passage of Sumi may seem to indicate that this "execution code" is an

²⁰ Present application, page 3, lines 8-14 ("The file, line-number approach is generally used in interactive application debuggers, but depends on the availability of relevant debugging information in the executable image. This approach requires the debugger itself to understand and use this knowledge, to relate the virtual address of the breakpoint to corresponding file name, line number when the breakpoint is hit. Some support from the OS is also needed to achieve this correlation. *Executable images are not generally built with debugging information.* Hence this method cannot be used to identify global breakpoints on generally available executable images.") (emphasis added); page 7, lines 21-25 ("As shown in Fig. 1, there are for example three processes A, B, and C to which a global breakpoint can be applied. The virtual address space of each process A, B, and C is shown. In each of the virtual address spaces, there is a code segment 102, 104, 106 for memory mapped region of *executable image file XYZ.SO.*") (emphasis added).

²¹ See, e.g., present application, page 7, line 31 through page 8, line 10 ("Using the inode and the offset, the Memory Manager module 120 of the operating system kernel locates the physical page where the breakpoint instruction is actually implanted . . . The embodiments have a number of advantages. In particular, the debugger is freed from needing to know anything about the executable format. The method is universally applicable to all types of executables and also applicable to certain executable contexts (for example, shared libraries), which can be loaded at different virtual addresses in different processes. Further, the method does not require any additional debugging information to be present in the executable file.").

²² Sumi, col. 3, lines 22-30.

²³ Sumi, col. 16, lines 49-64 ("The code generation unit 110 converts the program rewritten by the optimization unit 106 into execution code and stores the result in the generated code storage unit 103. Here, "execution code" refers to code which can be decoded and executed by the hardware of the target machine. Of special note here is the scale of the target machine, with the program conversion apparatus or the present embodiment generating execution for a system having the hardware model shown in FIG. 7. Here, the central processing unit (CPU) shown in FIG. 7 is fundamentally equipped with the data registers D0, D1, D2, and D3, the address registers A0, A1, A2, and A3, and an arithmetic logic unit (ALU). The CPU is additionally provided with a program counter PC which shows what address is currently being executed, and a stack pointer SP which expresses a starting address in a current stack. Here, the execution code and stack are located in memory.").

actual executable image residing in memory of a target computer system, it is clear from other disclosure by Sumi that this is not the case. Sumi sets out FIG. 8A to show an example of the “execution code” sequence stored by the generated code storage unit 103.²⁴ Sumi describes the meaning of the execution code “at each address” in FIG. 8A.²⁵ These addresses are locations in the generated code storage unit 103.²⁶ This generated code storage unit 103 is part of a larger apparatus disclosed by Sumi, a “program development system.”²⁷ The execution code in the generated code storage unit 103 is processed by a code execution unit 206.²⁸ This code execution unit 206 is *not* the target computer system, and, correspondingly, the “execution code” is not the executable image residing in the memory of the target system.²⁹

Note also that the execution code shown in FIG. 8A of Sumi is in the format of quasi-human-readable sort of machine-language assembly code. The addresses shown are locations of lines in an assembly code listing that include “x” in each “address.”³⁰ These x’s indicate that it is not known where the memory locations for the lines of a corresponding executable image of the file are actually located in the target system. This is because the focus of Sumi is on a tool to help a developer understand how a break point that the developer places in a

²⁴ Sumi, col. 8, lines 10-11.

²⁵ Sumi, col. 17, lines 15-16.

²⁶ Sumi, col. 10, lines 22-28. (“The generated code storage unit 103 is similarly divided into a plurality of small regions, with each of these small regions having a storage region which is assigned an address. These storage regions are used to store execution code when execution code has been generated by the code generation unit 110.”).

²⁷ Sumi, col. 21, lines 27-35 (“The entire construction of the program development system in the first embodiment is shown in FIG. 4. In this embodiment, the program development system is composed of a program conversion apparatus equipped with a debugging information generation apparatus, a debugging apparatus, a program storage unit 101, a primitive storage unit 102, a generated code storage unit 103, and a debugging information storage unit 104. Here, the information stored by the program storage unit 101, the primitive storage unit 102, the generated code storage unit 103, and the debugging information storage unit 104 is used by both the program conversion apparatus and the debugging apparatus.”).

²⁸ Sumi, col. 21, lines 27-35 (“The flowchart in FIG. 13 shows the details of the processing performed by the code execution unit 206 . . . the processing advances to step S72 where the first address “0.times.100” [sic] stored in the generated code storage unit 103 is set in the program counter. Next, in step S73, the code at the address stored in the program counter is fetched from the generated code storage unit 103 . . .”).

²⁹ Sumi, col. 21, lines 6-26 (“The code execution unit 206 may be composed of any of a simulator, an incircuit [sic] emulator, or a monitor, so that by using the features which are unique to simulators, incircuit emulators, or monitors, the hardware environment of the target machine can be recreated and a plurality of items of execution code stored in the generated code storage unit 103 can be successively executed in this hardware environment until a breakpoint set by the breakpoint setting unit 203 is reached. . . . while there are differences between the hardware environments of a simulator, an incircuit emulator, and a monitor (while the hardware environments for a monitor or incircuit emulator are the same or very close to that of the real machine, the hardware environment for a simulator is merely a model which is generated on a host computer) all three hardware environments have a common aspect in that they recreate the functions of the data registers D0, D1, D2, and D3, the address registers A0, A1, A2, and A3, the program counter PC, and the stack pointer SP which are shown in the hardware model of FIG. 7.”).

³⁰ See Sumi FIG. 8A.

line of source code maps to optimized code. That is, to help the developer understand this, the optimized "execution code" is displayed in a manner that the developer can read.³¹

From the above it should be recognized that Sumi does not supply the teaching that is lacking in Rosenberg about setting a break point in an executable. To make more clear the patentable distinction of the present invention, claim 1 and its counterpart independent claims, 14 and 27, were amended in response to the First Office Action. In particular, amended claim 1 states, by way of predicate, that a file name is received "for an executable image file," that "the executable image file is loaded in memory of a computer system for execution by the computer system" and that "the global breakpoint is to be placed in the image." Neither Sumi or Rosenberg teach or suggest this.

It should also be appreciated that the present invention concerns an aspect of bridging a gap between a user, who needs to be able to specify a break point in a manner that is easily intelligible to the user, and a debugger that needs to automatically insert, i.e., "represent," the break point in an executable image responsive to the information specified by the user. And it concerns doing this aspect of debugging in a way that is substantially universal, rather than by means of specialized debugger software. The present invention claims debugging the executable image by using operating system modules of a computer system that are the same modules available in the target computer system. That is, the modules are *not* special modules of debugger software. Accordingly, Claim 1 and its counterparts were previously amended to state that a symbol expression and the file name are passed "to a first operating system module running on the computer system" and that the symbol expression is "for a location in the executable image file where the global breakpoint is to be placed." Similarly, claim 1 and its counterparts were previously amended to state that the file name for the executable image file is passed to a second operating system module and that a file offset and file identifier for the executable image are returned by the modules and represented *in* the executable, i.e., inserted into the executable image file.

Thus the present invention is clearly distinguished from the teaching of Sumi in which execution code is processed by a simulator. In Sumi, the breakpoint setting unit 203 of the

³¹ Sumi, col. 5, lines 47-56 ("[The system] displays the analyzed details, so that even if the original source code statements have been greatly rewritten, the program developer will soon be able to understand how his/her program has been rewritten during optimization. By doing so, the program developer will not be baffled by the optimization of the program, and will be able to perform the operation verification of the machine language program while conscious of the source code statements written in the high-level programming language.").

development system receives a line indication for a break point from a user and converts this into an address in the execution code.³² There is no suggestion by Sumi that this unit 203 includes or uses an operating system module to determine an address, nor that the address that the unit 203 does determine is an actual memory location for an instruction of an executable image loaded in a target computer system.

The Final Office Action asserts that since Rosenberg teaches an executable image file, and that since a file inherently possesses a file name, therefore “it is easy” to lookup a symbol expression and/or offset for allocation in the executable image file by looking up the symbol tables of the file.³³ However easy this might or might not be, the Final Office Action does not cite any basis to support the contention.

An asserted inherent characteristic may be shown by extrinsic evidence, if the missing descriptive matter is necessarily present.³⁴ But the Final Office Action does not cite extrinsic evidence of the asserted inherent features. Moreover, the Final Office Action summarily assumes that it is an inherent element or function of Rosenberg’s teaching that a symbol expression and/or offset for allocation in an executable image file is looked up in symbol tables of the executable file, despite a lack of even a suggestion by Rosenberg of such an arrangement. But the asserted feature is not necessarily present. This is because the asserted feature is contrary to logic, since Rosenberg involves specifying a file name and line number or file offset in a source file, not in an executable, as the Final Office Action admits.

Aside from the above described differences between the cited teachings and that of the present invention, as claimed, the two techniques mentioned are well known for identifying breakpoints and they implicitly suffer from limitations that do not apply to the present invention. Rosenberg’s <filename, line number> method would only work if additional debugging information is available at run time. Sumi’s <virtual address> method would only work if an executable process were running and only one particular instance was under a debugger or an assumption is made at breakpoint insertion time as to load location of executable code, which will fail for shared executable objects. The limitations of these two techniques are well understood for a long time and people skilled in the art of debugging have learnt to operate under these constraints as nobody has come up with any method that overcomes the limitations of both

³² Col. 19, lines 11-15.

³³ Final Office Action, page 13, item 12.

³⁴ MPEP 2131.01, III (citing *Continental Can Co. USA v. Monsanto Co.*, 948 F.2d 1264, 1268).

these techniques. The present invention does not have these limitations. It provides a method of identifying a breakpoint that does not require additional debug information at run time and one that does not make any assumptions about load location or process contexts of executable code being debugged.³⁵

Claim 1 has been discussed at length above. However, the same discussion applies to independent claims 14 and 27, which have similar language. Applicant contends that the independent claims 1, 14 and 27 are patentably distinct since none of the rejections cite the use of operating system modules to determine actual memory locations for instructions of an executable image loaded in a target computer system and inserting these locations in the executable image.

Also, Applicant contends claims 2, 5, 7-13, 15, 18, 20-26, 28, 31 and 33-39 are allowable at least due to being dependent upon allowable claims.³⁶

Issue 2. The Final Office Action rejects claims 3, 16 and 29 on the basis that they are obvious in view of Rosenberg, Sumi and O'Connor. Appellant respectfully disagrees. While Appellant recognizes that the concept of using an inode to represent a file within the operating system is a standard practice and has been well documented, nevertheless, using the inode number in the representation of a breakpoint is not taught or suggested by the cited art. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination.³⁷ O'Connor does not concern break points or even debugging. The notion of an inode is typically a low-level operating system abstraction, so that to the best of Applicant's knowledge it has not occurred to those skilled in the art to use the inode for breakpoint representation despite its benefit of allowing globally shared breakpoints rather than process instance specific breakpoints. The mere mention of the inode function by O'Connor does not suggest the desirability of the combination of O'Connor with Rosenberg and Sumi, as claimed for the present invention.

Issue 3. The Final Office Action rejects claims 4, 17 and 30 on the basis that they are obvious in view of Rosenberg, Sumi and Starek. Appellant respectfully disagrees. While

³⁵ Present application, page 8, lines 5-12.

³⁶ MPEP 2143.03 ("If an independent claim is nonobvious under 35 U.S.C. 103, then any claim depending therefrom is nonobvious," citing *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed. Cir. 1988)).

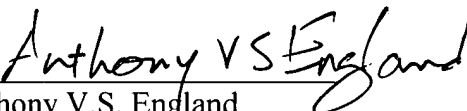
³⁷ MPEP 2143.01 (citing *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990)).

cited art.³⁸ Starek does not concern break points or even debugging. The notion of a File Control Block is typically a low-level operating system abstraction, so that to the best of Applicant's knowledge it has not occurred to those skilled in the art to use the File Control Block for breakpoint representation despite its benefit of allowing globally shared breakpoints rather than process instance specific breakpoints. The mere mention of the File Control Block function by Starek does not suggest the desirability of the combination of O'Connor with Rosenberg and Sumi, as claimed for the present invention.

REQUEST FOR ACTION

Based on the above arguments, Appellant requests that claims 1-5, 7-18, 20-31 and 33-39 of the present application be allowed and the application promptly be passed to issuance.

Respectfully submitted,

By 
Anthony V.S. England
Registration No. 35,129
Attorney of Record for
IBM Corporation
1717 West Sixth Street, Suite 230
Austin, Texas 78703
Telephone: 512-477-7165
a@aengland.com

ATTACHMENTS:

APPENDIX "AA" CLAIMS

³⁸ Id.

APPENDIX "AA"

What is claimed is:

1. (previously presented) A method of identifying a global breakpoint for debugging computer software, said method including the steps of:
 - receiving a file name for an executable image file, wherein the executable image file is loaded in memory of a computer system and the global breakpoint is to be placed in the image for executing by the computer system;
 - receiving a symbol expression for a location in the executable image file where the global breakpoint is to be placed;
 - passing the symbol expression and the file name to a first operating system module running on the computer system;
 - receiving a file offset corresponding to the symbol location from the first operating system module;
 - passing the file name for the executable image file to a second operating system module;
 - receiving a file identifier from the second operating system module, wherein the file identifier is used by the operating system for uniquely identifying the executable file in the computer system memory; and
 - representing said global breakpoint in code of said software using the received file identifier of the executable image file and the received offset in said executable file.
2. (original) The method according to claim 1, wherein said file identifier is a file name.
3. (previously presented) The method according to claim 1, wherein said file identifier is an inode of a Unix operating system.
4. (original) The method according to claim 1, wherein said file identifier is a file control block of a non-Unix operating system.
5. (original) The method according to claim 1, further including the step of resolving a virtual address of said code to said file identifier and said offset.

6. (canceled)
7. (original) The method according to claim 1, further including the step of providing a hash list to look up said global breakpoint using said file identifier and said offset.
8. (original) The method according to claim 7, further including the step of maintaining said hash list of global breakpoints based on file identifiers and offsets.
9. (original) The method according to claim 1, further including the step of deriving said file identifier and said offset using a virtual address.
10. (original) The method according to claim 9, wherein said deriving step is dependent upon information maintained by an operating system to map executable files to memory.
11. (original) The method according to claim 9, further including the step of:
determining file identifier and said offset from said virtual address for a memory mapped region.
12. (original) The method according to claim 9, wherein two or more virtual addresses exist for said software code.
13. (original) The method according to claim 1, wherein said global breakpoint is contained in a private-per-process copy of a physical page of said software code.

14. (previously presented) A computer-implemented apparatus for identifying a global breakpoint for debugging computer software, said apparatus including:

- a central processing unit for executing said computer software;
- memory for storing at least a portion of said computer software;
- means for receiving a file name for an executable image file, wherein the executable image file is loaded in memory of a computer system and the global breakpoint is to be placed in the image for executing by the computer system;
- means for receiving a symbol expression for a location in the executable image file where the global breakpoint is to be placed;
- means for passing the symbol expression and the file name to a first operating system module running on the computer system;
- means for receiving a file offset corresponding to the symbol location from the first operating system module;
- means for passing the file name for the executable image file to a second operating system module;
- means for receiving a file identifier from the second operating system module, wherein the file identifier is used by the operating system for uniquely identifying the executable file in the computer system memory; and
- means for representing said global breakpoint in code of said computer software using the received file identifier of the executable image file and the received offset in said executable file.

15. (original) The apparatus according to claim 14, wherein said file identifier is a file name.

16. (previously presented) The apparatus according to claim 14, wherein said file identifier is an inode of a Unix operating system.

17. (original) The apparatus according to claim 14, wherein said file identifier is a file control block of a non-Unix operating system.

18. (original) The apparatus according to claim 14, further including means for resolving

a virtual address of said code to said file identifier and said offset.

19. (canceled)

20. (original) The apparatus according to claim 14, further including a hash list to look up said global breakpoint using said file identifier and said offset.

21. (original) The apparatus according to claim 20, further including means for maintaining said hash list of global breakpoints based on file identifiers and offsets.

22. (original) The apparatus according to claim 14, further including means for deriving said file identifier and said offset using a virtual address.

23. (original) The apparatus according to claim 22, wherein said deriving means is dependent upon information maintained by an operating system to map executable files to memory.

24. (original) The apparatus according to claim 22, further including means for determining said file identifier and said offset from said virtual memory address for a memory mapped region.

25. (original) The apparatus according to claim 22, wherein two or more virtual addresses exist for said software code.

26. (original) The apparatus according to claim 14, wherein said global breakpoint is contained in a private-per-process copy of a physical page of said software code.

27. (previously presented) A computer program product having a computer readable medium having a computer program recorded therein for identifying a global breakpoint for debugging computer software, said computer program product including:

computer program means for receiving a file name for an executable image file, wherein the executable image file is loaded in memory of a computer system and the global breakpoint is to be placed in the image for executing by the computer system;

computer program means for receiving a symbol expression for a location in the executable image file where the global breakpoint is to be placed;

computer program means for passing the symbol expression and the file name to a first operating system module running on the computer system;

computer program means for receiving a file offset corresponding to the symbol location from the first operating system module;

computer program means for passing the file name for the executable image file to a second operating system module;

computer program means for receiving a file identifier from the second operating system module, wherein the file identifier is used by the operating system for uniquely identifying the executable file in the computer system memory; and

computer program code means for representing said global breakpoint in code of said computer software using the received file identifier of the executable image file and the received offset in said executable file.

28. (original) The computer program product according to claim 27, wherein said file identifier is a file name.

29. (previously presented) The computer program product according to claim 27, wherein said file identifier is an inode of a Unix operating system.

30. (original) The computer program product according to claim 27, wherein said file identifier is a file control block of a non-Unix operating system.

31. (original) The computer program product according to claim 27, further including

computer program code means for resolving a virtual address of said code to said file identifier and said offset.

32. (canceled)

33. (original) The computer program product according to claim 27, further including computer program code means for providing a hash list to look up said global breakpoint using said file identifier and said offset.

34. (original) The computer program product according to claim 33, further including computer program code means for maintaining said hash list of global breakpoints based on file identifiers and offsets.

35. (original) The computer program product according to claim 27, further including computer program code means for deriving said file identifier and said offset using a virtual address.

36. (original) The computer program product according to claim 35, wherein said computer program code means for deriving is dependent upon information maintained by an operating system to map executable files to memory.

37. (original) The computer program product according to claim 35, further including computer program code means for determining said file identifier and said offset from said virtual memory address for a memory mapped region.

38. (original) The computer program product according to claim 35, wherein two or more virtual addresses exist for said software code.

39. (original) The computer program product according to claim 27, wherein said global breakpoint is contained in a private-per-process copy of a physical page of said software code.